8|8|21 Functions/Methods in JAVA

Functions/Methods (in java):

- A method is a block of code which only runs when it is called.
- To reuse code: define the code once, & use it many times.

Syntax:

```
public class Main {
        static void myMethod() {
                // code
        }
}
```

void → this method myMethod() does not have a return value.

myMethod → name of method

```
public class Main {
        access_modifier return_type method(){
                // code
                return statement;
        }
}
```

return statement; → f^n ends here

- method() → calling the function.

method → name of function

- **return_type :-**

A return statement causes the program control to transfer back to the caller of a method.
A return type may be premitive type like int, char, or void type (returns nothing).

⇒ there are a few important things to understand about returning the values:

- The type of data returned by a method must be compatible with the return type specified by the method.
  eg: if return type of some method is boolean, we cannot return an integer.
- The variable recieving the value returned by a method must also be compatible with the return type specified for the method.

⇒ Pass by value:

eg 1:

object/value

```
main() {
    name = "a";
    greet(name);
}
static void greet(naam) {
    print(naam)
}
...
```

Creating copy of value of name

i.e., passing value of the reference.

$$\begin{tikzcd} \text{name} \arrow[r] & \boxed{a} \\ \text{naam} \arrow[ur] & \end{tikzcd}$$

eg2:

```
psvm() {
    name = "a";
    change(name);
    print(name);
}
change(naam){
    naam = "b";
}
```

Creating copy

$$\begin{tikzcd} \text{name} \arrow[r] & \boxed{a} \\ \text{naam} \arrow[ur] & \end{tikzcd}$$

$$\begin{tikzcd} \text{name} \arrow[r] & \boxed{a} \\ \text{naam} \arrow[r] & \boxed{b} \end{tikzcd}$$

{ not changing original object, just creating new object.

since it is created inside fn it will not change original one.

✱ <u>points to be noted :</u>

1→• <u>primitive data type</u> like int, short, char, byte etc.
↳ **just pass value**

2→• <u>object & reference</u> :
↳ **passing value of reference variable.**

<u>eg-1 :</u>

```
psvm() {
      a=10;
      b=20;
      swap(a,b);
}
swap(num1, num2) {
        temp = num1;
        num1 = num2;
        num2 = temp;
}
```

a → 10
b → 20
] but not here

temp → 10
num1 → 20
num2 → 10
] at fn scope level they are swapped.

Here, they just passes the value....

<u>eg-2 :</u>

$$\begin{tikzcd}
\text{arr} \arrow[r] & {[1, 2, 3, 4, 5]} \\
\text{nums} \arrow[ur]
\end{tikzcd}$$

nums[0] = 99 [now, the value of 0th position in nums will change which also changes value of arr[0]]

$$\begin{tikzcd}
\text{arr} \arrow[r] & {[99, 2, 3, 4, 5]} \\
\text{nums} \arrow[ur]
\end{tikzcd}$$

Here, passing value of reference variable

✱ Scopes:

• **function scope :**
variables declared inside a method / function scope (means inside method) can't be accessed outside the method.

[illegible] eg:-

```
psvm () {
                ✗
}
all () {
  (int x);   → can't be accessed outside
}
```

• **block scope:**

```
psvm ( ) {
      int a = 10;      ]
      int b = 20;      ]  → variables initialized outside the block
                             can be updated inside the box.
    {  int a = 5; ✗
         a = 100; ✓    ]
         int c = 20;   ]  → variables initialized inside the block
    }                        cannot be updated outside the box but
    c = 10; ✗                can be reinitialized outside the block.
    int c = 15; ✓
    a = 50; ✓
}
```

variables like "a" here, is declared outside the block, updated inside the block and can also be updated outside the block.

• **loop scope :**
variables declared inside loop [illegible] are having loop scope.

⇒ <u>Shadowing</u>:

Shadowing in Java is the practice of using variables in overlapping scopes with the same name where the variable in low-level scope overrides the variable of high-level scope. Here the variable at high-level scope is shadowed by low-level scope variable.

eg:-
```
public class Shadowing {
        static int x = 90;
        psvm ( ) {.
                System.out.println (x);        → 90
                x = 50;          // here high-level scope is shadowed by low-level scope
                System.out.println (x);        → 50
        }
}
```

⇒ <u>Variable Arguments</u>:

. Variable Arguments is used to take a variable number of arguments. A method that takes a variable number of arguments is a varargs method.

<u>Syntax</u>:
```
        static void fun (int ...a) {
                    // method body
        }
```

Here, parameters would be array of type int [ ] method)

⇒ Function Overloading:

Function Overloading happens when two functions have same name.

eg → 1)
```
fun ( ) {
    //code
}

fun ( ) {
    //code
}
```
× function overloading

2)
```
fun (int a) {
        //code
    }

fun (int a, intb) {
        //code
    }
```
This is allowed having different arguments with same method name.

⇒ At compile time, it decides which fⁿ to run.

⇒ Armstrong number:

suppose there is number → 153

$153 \longrightarrow (1)^3+(5)^3+(3)^3 = 1+125+27$

$= \underline{\underline{153}}$